```
//@version=5
indicator(title="Nebula", shorttitle="Nebula v2.2", overlay=true, max_bars_back=1000, max_lines_count=

// This source code is subject to the terms of the Mozilla Public License 2.0 at https://mozilla.org/M

// Nebula was created by TraderOracle (DaveTrade55 on TradingView).  Please watch my shitty YouTube ch

// Nebula was inspired by Daviddtech's year end video on "best indicators" here: https://www.youtube.c
// My thoughts were - why not combine them all into a pretty cloud, since two are EMAs?  Thus Nebula w
// I then added all my TO Method reversal indicators, and added Aaron D's Wave theory idea
// ( see Aaron D's channel here: https://www.youtube.com/watch?v=blg8GCxR80o )

// @davidclarke6612 provided the idea for Rational Quadratic Kernel as a replacement for the 9/21 cros

const int vSqueeze = 4
const int vTramp = 4
const int vBands = 2
const int vRSI = 2
const int vLuxRev = 3
const int vEarlyRev = 2
const int vDeadRev = 2
const int vShark = 2

I_Like_Big_Butts_And_I_Cannot_Lie = input.bool(false, title="NOTE: Uncheck body/wick/border in your Se

cloudType = input.string(title="Cloud Type: ", options=["None", "Simple", "Relative Strength", "Money
sCandleType = input.string(title="Candle Coloring: ", options=["None", "Vector", "Waddah", "Squeeze",
Theme = input.string(title="Color Theme: ", options=["Standard", "Pinky and the Brain", "Color Blind",

bShowHEMA = input.bool(false, "Show HEMA line", group="Visible Settings")
bShowPlus = input.bool(true, "Show plus sign to add", group="Visible Settings")
bShowBigPlus = input.bool(true, "Show bigger plus sign (Vodka Shot)", group="Visible Settings")
bEnhance1 = input.bool(true, "Enhance #1 if Buy/Sell signal nearby", group="Visible Settings")
bShowProfit = input.bool(true, "Show take profit suggestions", group="Visible Settings")
bShow921 = input.bool(false, "Show 9/21 EMA cross", group="Visible Settings")

bIgnoreDoji = input.bool(false, "Ignore dojis (only use on NQ, 1 min)", group="Basic Settings")
bNoCounterTrend = input.bool(false, "Don't show counter trend trades (only use on NQ, 1 min)", group="
iProfit = input.int(5, "Minimum signals for take partial profit", minval=2, maxval=50, group="Basic Se
iMaxProfit = input.int(7, "Minimum signals for take ALL profit", minval=3, maxval=50, group="Basic Set
iMaxBody = input.int(1, "Body size to consider a doji", minval=1, maxval=4, group="Basic Settings")

//cfgBrightness = input.int(25, "Candle brightness for a blank candle", minval=1, maxval=100, group="B
//cfgBrightCloud = input.int(60, "Candle brightness for opposite cloud color signal", minval=1, maxval
//iMinCloud = input.int(2, "Minimum cloud width to signal in", minval=2, maxval=30, group="Basic Setti

//sTidalWave = input.string(title="Tidal Wave Alerts: ", options=["None", "First Candle", "First Color
//STPercent = input.int(44, "Candle brightness for [First Value Candle] option (1 to 99)", group="Tida

adxSqueeze = input.int(0, title="ADX Threshold for Tidal Wave", group="Tidal Wave Settings", tooltip="
bTrackBar = input.bool(false, "Show volume imbalances", group="Tidal Wave Settings")
iBarExtend = input.int(50, "Number of bars to extend line", maxval=500, minval=10, group="Tidal Wave S

```

```
lWidth = input.int(3, "Line Width", group="Tidal Wave Settings")
lStyle = input.string(title="Line Style", options=["Solid", "Dotted", "Dashed"], defval="Dotted", grou

sStyle = lStyle=="Solid" ? line.style_solid : lStyle=="Dashed" ? line.style_dashed : line.style_dottec
bExtend = lWidth==500 ? extend.right : extend.none

bShowRevPattern = input.bool(false, "Outline reversal candle patterns in yellow", group="Advanced")
bShowRetest = input.bool(false, "Show high/low retests", group="Advanced")
bShowQuad = input.bool(false, "Use quadratic equation for 9/21 cross", group="Advanced")
iSimpleCloud = input.int(80, title="Simple Cloud Opacity (0=brightest, 100=invisible)", group="Advance
iLowCloud = input.int(80, title="Cloud Opacity Lower Limit (0=brightest, 100=invisible)", group="Advar
iHighCloud = input.int(50, title="Cloud Opacity Upper Limit (0=brightest, 100=invisible)", group="Adva
iTopBody = input.int(80, title="Top WAE Body Value", group="Advanced")
iTopBorder = input.int(33, title="Top WAE Border Value", group="Advanced")
iVolDeltaTop = input.int(300, title="Cumulative Volume Delta top end", group="Advanced")

ADX_Length = input.int(2, title="ADX_Length", group="Fantail VMA")
Weighting = input.float(10.0, title="Weighting", group="Fantail VMA")
MA_Length = input.int(6, minval=1, title="MA_Length", group="Fantail VMA")

colorBigGreen = Theme == "Standard" ? color.new(#00ff00, 0) : Theme == "Pinky and the Brain" ? color.r
colorBigRed = Theme == "Standard" ? color.new(#ff0000, 0) : Theme == "Pinky and the Brain" ? color.new

float iSource = 0.0
float MIN_CLOUD = 0.0
float MID_CLOUD = 0.0
float MAX_CLOUD = 0.0
var isLong78 = false
var isShort78 = false

iTPSignalCount = 0
ema9 = ta.ema(close, 9)
ema21 = ta.ema(close, 21)
bodySize = math.abs(close - open)

adxlen = input(14, title="ADX Smoothing", group="ADX")
dilen = input(14, title="DI Length", group="ADX")
dirmov(len) =>
    up5 = ta.change(high)
    down5 = -ta.change(low)
    plusDM = na(up5) ? na : (up5 > down5 and up5 > 0 ? up5 : 0)
    minusDM = na(down5) ? na : (down5 > up5 and down5 > 0 ? down5 : 0)
    truerange = ta.rma(ta.tr, len)
    plus = fixnan(100 * ta.rma(plusDM, len) / truerange)
    minus = fixnan(100 * ta.rma(minusDM, len) / truerange)
    [plus, minus]
adx(dilen, adxlen) =>
    [plus, minus] = dirmov(dilen)
    sum = plus + minus
    adx = 100 * ta.rma(math.abs(plus - minus) / (sum == 0 ? 1 : sum), adxlen)
adxValue = adx(dilen, adxlen)
sigabove19 = adxValue > adxSqueeze

var arLabel = array.new_label(0)
f_AddChar(BarIndex, UpDown, Color, Char, Ticks, Size) =>
```

```
    if (UpDown=="Up")
        array.push(arLabel, label.new(x=bar_index, y=na, style=label.style_none, color=color.new(color
    else
        array.push(arLabel, label.new(x=bar_index, y=na, style=label.style_none, color=color.new(color
    if array.size(arLabel) >= 498
        ln = array.shift(arLabel)
        label.delete(ln)


// =-=-=-=-=-=-=-=-=-=-=-=-=   Cumulative Volume Delta © Ankit_1618   =-=-=-=-=-=-=-=-=-=-=-=-= //

upper_wick = close>open ? high-close : high-open
lower_wick = close>open ? open-low : close-low
spread = high-low
body_length = spread - (upper_wick + lower_wick)

percent_upper_wick = upper_wick/spread
percent_lower_wick = lower_wick/spread
percent_body_length = body_length/spread

buying_volume = close > open ? (percent_body_length + (percent_upper_wick + percent_lower_wick)/2)*vol
selling_volume = close < open ? (percent_body_length + (percent_upper_wick + percent_lower_wick)/2)*vc

cumulative_buying_volume = ta.ema(buying_volume,14)
cumulative_selling_volume = ta.ema(selling_volume,14)

cumulative_volume_delta = cumulative_buying_volume - cumulative_selling_volume
//plot(cumulative_volume_delta, color= cumulative_volume_delta>0 ? color.green : color.red, style=plot

cCVDColor = color.white
cCVDBorder = color.white
cCVDWick = color.white

if cumulative_volume_delta > 0
    cCVDColor := color.from_gradient(math.abs(cumulative_volume_delta), 0, iVolDeltaTop, color.new(col
    cCVDBorder := cCVDColor
    cCVDWick := cCVDColor
else if cumulative_volume_delta < 0
    cCVDColor := color.from_gradient(math.abs(cumulative_volume_delta), 0, iVolDeltaTop, color.new(col
    cCVDBorder := cCVDColor
    cCVDWick := cCVDColor


// =-=-=-=-=-=-=-=-=-=-=-=-=   Dead Simple Reversal   =-=-=-=-=-=-=-=-=-=-=-=-= //

c1 = close[1] < open[1] and close > open
c2 = close > open[1]
c3 = ta.lowest(low,3) < ta.lowest(low,50)[1] or ta.lowest(low,3) < ta.lowest(low,50)[2] or ta.lowest(l
buyDSR = c1 and c2 and c3

c4 = close[1] > open[1] and close < open
c5 = close < open[1]
c6 = ta.highest(high,3) > ta.highest(high,50)[1] or ta.highest(high,3) > ta.highest(high,50)[2] or ta.
sellDSR = c4 and c5 and c6
```

```
if (buyDSR or sellDSR) or (buyDSR[1] or sellDSR[1])
    iTPSignalCount := iTPSignalCount + vDeadRev

//plotshape(bDSR ? buyDSR : na, location=location.belowbar, style=shape.square, size=size.tiny)
//plotshape(bDSR ? sellDSR : na, location=location.abovebar, style=shape.square, size=size.tiny)


// =-=-=-=-=-=-=-=-=-=-=-=-=  Reversal Signals [LuxAlgo]  =-=-=-=-=-=-=-=-=-=-=-=-= //
bSh  = 'Completed'
ptLT = 'Step Line w/ Diamonds'
ptSR = 'Circles'
eSh  = 'Completed'

Bcmpltd    = bSh == 'Completed'

var noShw = false
cmpltd    = eSh == 'Completed'
noShw    := eSh == 'None' ? false : true

type bar
    float o = open
    float h = high
    float l = low
    float c = close
    int   i = bar_index

type trb
    int   bSC
    float bSH
    float bSL

    int   sSC
    float sSH
    float sSL

type tre
    int   bCC
    float bC8
    float bCHt
    float bCH
    float bCL
    float bCLt
    float bCD

    int   sCC
    float sC8
    float sCHt
    float sCH
    float sCL
    float sCLt
    float sCT

bar b = bar.new()
var trb S = trb.new()
var tre C = tre.new()
```

```
noC  = #00000000
rdC  = #f23645
gnC  = #089981
whC  = #ffffff
blC  = #2962ff
grC  = #787b86
bgC  = #00bcd4

shpD = shape.labeldown
shpU = shape.labelup
locA = location.abovebar
locB = location.belowbar
dspN = false
pltL = plot.style_circles
pltS = size.tiny

f_xLX(_p, _l) =>
    (_l > _p and _l < _p[1]) or (_l < _p and _l > _p[1])

f_lnS(_s) =>
    s = switch _s
        'Circles'               => plot.style_circles
        'Step Line'             => plot.style_steplinebr
        'Step Line w/ Diamonds' => plot.style_steplinebr

ptLB = f_lnS(ptLT)
ptRS = f_lnS(ptSR)
con = b.c < b.c[4]

if con
    S.bSC := S.bSC == 9 ? 1 : S.bSC + 1
    S.sSC := 0
else
    S.sSC := S.sSC == 9 ? 1 : S.sSC + 1
    S.bSC := 0

pbS = (b.l <= b.l[3] and b.l <= b.l[2]) or (b.l[1] <= b.l[3] and b.l[1] <= b.l[2])

bShowUppies = ((S.bSC == 9 and not pbS) or (S.bSC == 9 and pbS) or (S.bSC[1] == 8 and S.sSC == 1)) and
//plotshape(bShowUppies ? 1 : na, title="Reversal Approaching", style=shape.xcross, location=location.

bC8  = S.bSC[1] == 8 and S.sSC == 1
sR   = ta.highest(9)
bSR  = 0.0
bSR := S.bSC == 9 or bC8 ? sR : b.c > bSR[1] ? 0 : bSR[1]

if S.bSC == 1
    S.bSL := b.l

if S.bSC > 0
    S.bSL := math.min(b.l, S.bSL)

    if b.l == S.bSL
        S.bSH := b.h
```

```
bSD  = 0.0
bSD := S.bSC == 9 ? 2 * S.bSL - S.bSH : b.c < bSD[1] or S.sSC == 9 ? 0 : bSD[1]
psS = (b.h >= b.h[3] and b.h >= b.h[2]) or (b.h[1] >= b.h[3] and b.h[1] >= b.h[2])

//if (S.sSC == 9 and not psS) or (S.sSC == 9 and psS) or (S.sSC[1] == 8 and S.bSC == 1) and BnoShw and
//    f_AddCharStd(bar_index, "Down", colorBigRed, label.style_xcross, 2)

bShowDownies = ((S.sSC == 9 and not psS) or (S.sSC == 9 and psS) or (S.sSC[1] == 8 and S.bSC == 1)) ar
//plotshape(bShowDownies  ? 1 : na, title="Reversal Approaching", style=shape.xcross, location=locatic

if (bShowUppies or bShowDownies) or (bShowUppies[1] or bShowDownies[1])
    iTPSignalCount := iTPSignalCount + vLuxRev


// =-=-=-=-=-=-=-=-=-=-=-=  LuxAlgo - Market Structure (Fractal)  =-=-=-=-=-=-=-=-=-=-=-=-= //

var float upOpen = na
var float upClose = na
var float downOpen = na
var float downClose = na

bGreenSignal = false
bRedSignal = false

length = 5 // default, min 3

type fractal
    float value
    int loc
    bool iscrossed

var pT = int(length / 2)
n = bar_index
dhT = math.sum(math.sign(high - high[1]), pT)
dlT = math.sum(math.sign(low - low[1]), pT)
bullf = dhT == -pT and dhT[pT] == pT and high[pT] == ta.highest(length)
bearf = dlT == pT and dlT[pT] == -pT and low[pT] == ta.lowest(length)
bullf_count = ta.cum(bullf ? 1 : 0)
bearf_count = ta.cum(bearf ? 1 : 0)

var upperT = fractal.new()
var line lower_lvl = na
var label ms_lbl = na
var bull_ms_count = 0
var broken_sup = false
var os = 0

if bullf
    upperT.value := high[pT]
    upperT.loc := n-pT
    upperT.iscrossed := false

if ta.crossover(close, upperT.value) and not upperT.iscrossed
    upOpen := open
```

```
    upClose := close
else if not broken_sup
    lower_lvl.set_x2(n)
    if close < lower_lvl.get_y2()
        broken_sup := true

var lowerT = fractal.new()
var line upper_lvl = na
var broken_res = false
var bear_ms_count = 0

if bearf
    lowerT.value := low[pT]
    lowerT.loc := n-pT
    lowerT.iscrossed := false

if ta.crossunder(close, lowerT.value) and not lowerT.iscrossed
    downOpen := open
    downClose := close
else if not broken_res
    upper_lvl.set_x2(n)
    if close > upper_lvl.get_y2()
        broken_res := true



// ==========================   MACD   ================================

fast_ma = request.security(syminfo.tickerid, "", ta.ema(close, 12))
slow_ma = request.security(syminfo.tickerid, "", ta.ema(close, 26))
macd = fast_ma - slow_ma
signal = request.security(syminfo.tickerid, "", ta.ema(macd, 9))

float hist = macd - signal
trend_up   = macd > signal
trend_dn   = macd < signal
cross_UP   = signal[1] >= macd[1] and signal < macd
cross_DN   = signal[1] <= macd[1] and signal > macd
cross_UP_A = (signal[1] >= macd[1] and signal < macd) and macd > 0
cross_DN_B = (signal[1] <= macd[1] and signal > macd) and macd < 0
//trend_col = trend_up ? col_trnd_Up : trend_up ? col_macd : show_trend  and trend_dn ? col_trnd_Dn: t

var bool histA_IsUp = false
var bool histA_IsDown = false
var bool histB_IsDown = false
var bool histB_IsUp = false
histA_IsUp   := hist == hist[1] ? histA_IsUp[1] : hist > hist[1] and hist > 0
histA_IsDown := hist == hist[1] ? histA_IsDown[1] : hist < hist[1] and hist > 0
histB_IsDown := hist == hist[1] ? histB_IsDown[1] : hist < hist[1] and hist <= 0
histB_IsUp   := hist == hist[1] ? histB_IsUp[1] : hist > hist[1] and hist <= 0
//hist_col =  histA_IsUp ? col_grow_above : histA_IsDown ? col_fall_above : histB_IsDown ? col_grow_be

cMDColor = color.white
cMDBorder = color.white
cMDWick = color.white
```

```
if histA_IsUp
    cMDColor := color.from_gradient(math.abs(hist), 0, 3, color.new(colorBigGreen, 30), color.new(colc
    cMDBorder := colorBigGreen
    cMDWick := cMDColor
else if histA_IsDown
    cMDColor := color.from_gradient(math.abs(hist), 0, 3, color.new(colorBigGreen, 80), color.new(colc
    cMDBorder := colorBigGreen
    cMDWick := cMDColor

if histB_IsUp
    cMDColor := color.from_gradient(math.abs(hist), 0, 4, color.new(colorBigRed, 30), color.new(colorE
    cMDBorder := colorBigRed
    cMDWick := cMDColor
else if histB_IsDown
    cMDColor := color.from_gradient(math.abs(hist), 0, 4, color.new(colorBigRed, 80), color.new(colorE
    cMDBorder := colorBigRed
    cMDWick := cMDColor

// plotcandle(sCandleType=="MACD" ? open : na, high, low, close, color=cMDColor, wickcolor=cMDWick, bc


// ==========================   Bixord: FantailVMA   ================================

rsi = ta.rsi(close, 14)
mfi = ta.mfi(hlc3, 14)
maCCI = ta.sma(hlc3, 20)
cci = (hlc3 - maCCI) / (0.015 * ta.dev(hlc3, 20))

if cloudType=="Relative Strength"
    iSource := rsi
    MIN_CLOUD := 20
    MID_CLOUD := 50
    MAX_CLOUD := 80

if cloudType=="Money Flow"
    iSource := mfi
    MIN_CLOUD := 20
    MID_CLOUD := 50
    MAX_CLOUD := 80

if cloudType=="Commodity Channel"
    iSource := cci
    MIN_CLOUD := 20
    MID_CLOUD := -100
    MAX_CLOUD := 100

VMA=close
VarMA=close
MA=close
STR = high-low
sPDI = 0.0
sMDI = 0.0
ADX=0.0
ADXR=0.0
```

```
Hi  = high
Hi1 = high[1]
Lo  = low
Lo1 = low[1]
Close1= close[1]

Bulls1 = 0.5*(math.abs(Hi-Hi1)+(Hi-Hi1))
Bears1 = 0.5*(math.abs(Lo1-Lo)+(Lo1-Lo))
Bears = Bulls1 > Bears1 ? 0 : (Bulls1 == Bears1 ? 0 : Bears1)
Bulls = Bulls1 < Bears1 ? 0 : (Bulls1 == Bears1 ? 0 : Bulls1)

if (bar_index > 0)
    sPDI := (Weighting*sPDI[1] + Bulls)/(Weighting+1)
    sMDI := (Weighting*sMDI[1] + Bears)/(Weighting+1)

TR = math.max(Hi-Lo,Hi-Close1)
if (bar_index > 0)
    STR  := (Weighting*STR[1] + TR)/(Weighting+1)

PDI = STR > 0 ? sPDI/STR : 0
MDI = STR > 0 ? sMDI/STR: 0
DX = (PDI + MDI) > 0 ? math.abs(PDI - MDI)/(PDI + MDI) : 0
if (bar_index > 0)
    ADX := (Weighting*ADX[1] + DX)/(Weighting+1)
vADX = ADX

adxlow = ta.lowest(ADX, ADX_Length)
adxmax = ta.highest(ADX, ADX_Length)
ADXmin = math.min(1000000.0, adxlow)
ADXmax = math.max(-1.0, adxmax)
Diff = ADXmax - ADXmin
Const = Diff > 0 ? (vADX- ADXmin)/Diff : 0

if (bar_index > 0)
    VarMA:=((2-Const)*VarMA[1]+Const*close)/2

FanVMA = ta.sma(VarMA,MA_Length)


// ==========================   McGinley Dynamic   ================================
mg = 0.0
mg := na(mg[1]) ? ta.ema(close, 14) : mg[1] + (close - mg[1]) / (14 * math.pow(close/mg[1], 4))

bCloudGreen = FanVMA > mg
bCloudRed = not bCloudGreen

// ==========================   Waddah Attar Explosion v1 by LazyBear   ==============================

sensitivity = input.int(150, title="Sensitivity", group="WAE")
fastLength = input.int(20, title="FastEMA Length", group="WAE")
slowLength = input.int(40, title="SlowEMA Length", group="WAE")
channelLength = input.int(20, title="BB Channel Length", group="WAE")
multWAE = input.float(2.0, title="BB Stdev Multiplier", group="WAE")

calc_macd(source, fastLength, slowLength) =>
```

```
    fastMA = ta.ema(source, fastLength)
    slowMA = ta.ema(source, slowLength)
    fastMA - slowMA

calc_BBUpper(source, length, mult) =>
    basis = ta.sma(source, length)
    dev = mult * ta.stdev(source, length)
    basis + dev

calc_BBLower(source, length, mult) =>
    basis = ta.sma(source, length)
    dev = mult * ta.stdev(source, length)
    basis - dev

upper = calc_BBUpper(close, channelLength, multWAE)
lower = calc_BBLower(close, channelLength, multWAE)

t1 = (calc_macd(close, fastLength, slowLength) - calc_macd(close[1], fastLength, slowLength))*sensitiv
e1 = (upper - lower)

trendUpWAE = (t1 >= 0) ? t1 : 0
trendDownWAE = (t1 < 0) ? (-1*t1) : 0

iCandleTrans = trendUpWAE > 0 ? math.round(math.abs(trendUpWAE - e1)) : math.round(math.abs(trendDownW

cBodyColor = color.white
cBorderColor = color.white
cWickColor = color.white

if (trendUpWAE > e1 and trendUpWAE > 0)
    cBodyColor := color.from_gradient(math.abs(trendUpWAE - e1), 1, iTopBody, color.new(colorBigGreen,
    cBorderColor := color.from_gradient(math.abs(trendUpWAE - e1), 1, iTopBorder, color.new(colorBigGr
    cWickColor := color.new(colorBigGreen, 0)

if (trendUpWAE < e1 and trendUpWAE > 0)
    cBodyColor := color.new(colorBigGreen, 90)
    cBorderColor := color.from_gradient(math.abs(e1 - trendUpWAE), 1, iTopBody, color.new(colorBigGree
    cWickColor := color.new(colorBigGreen, 30)

if (trendDownWAE > e1 and trendDownWAE > 0)
    cBodyColor := color.from_gradient(math.abs(trendDownWAE - e1), 1, iTopBody, color.new(colorBigRed,
    cBorderColor := color.from_gradient(math.abs(trendDownWAE - e1), 1, iTopBorder, color.new(colorBig
    cWickColor := color.new(colorBigRed, 0)

if (trendDownWAE < e1 and trendDownWAE > 0)
    cBodyColor := color.new(colorBigRed, 90)
    cBorderColor := color.from_gradient(math.abs(e1 - trendDownWAE), 1, iTopBody, color.new(colorBigRe
    cWickColor := color.new(colorBigRed, 30)



// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=   TRAMPOLINE  =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

// Idea from "Serious Backtester" - https://www.youtube.com/watch?v=2hX7qTamOAQ
// Defaults are optimized for 30 min candles
```

```
iBBThreshold = input.float(0.0015, minval=0.0, title="Bollinger Lower Threshold", tooltip="0.003 for c
RSIThreshold = input.int(25, minval=1, title="RSI Lower Threshold", tooltip="Normally 25", group="TRAM
RSIDown = input.int(72, minval=1, title="RSI Upper Threshold", tooltip="Normally 75", group="TRAMPOLIN

rsiLengthInput = input.int(14, minval=1, title="RSI Length", group="TRAMPOLINE Settings")
rsiSourceInput = input(close, "RSI Source", group="TRAMPOLINE Settings")
lengthBB = input.int(20, minval=1, group="TRAMPOLINE Bollinger Bands")
srcBB = input(close, title="Source", group="TRAMPOLINE Bollinger Bands")
multBB = input.float(2.0, minval=0.001, maxval=50, title="StdDev", group="TRAMPOLINE Bollinger Bands")
offsetBB = input.int(0, "Offset", minval = -500, maxval = 500, group="TRAMPOLINE Bollinger Bands")

isRed = close < open
isGreen = close > open

basisBB = ta.sma(srcBB, lengthBB)
devBB = multBB * ta.stdev(srcBB, lengthBB)
upperBB = basisBB + devBB
lowerBB = basisBB - devBB
downBB = low < lowerBB or high < lowerBB
upBB = low > upperBB or high > upperBB
bbw = (upperBB - lowerBB) / basisBB

up = ta.rma(math.max(ta.change(rsiSourceInput), 0), rsiLengthInput)
down = ta.rma(-math.min(ta.change(rsiSourceInput), 0), rsiLengthInput)
rsiM = down == 0 ? 100 : up == 0 ? 0 : 100 - (100 / (1 + up / down))

back1 = isRed[1] and rsiM[1] <= RSIThreshold and close[1] < lowerBB[1] and bbw[1] > iBBThreshold
back2 = isRed[2] and rsiM[2] <= RSIThreshold and close[2] < lowerBB[2] and bbw[2] > iBBThreshold
back3 = isRed[3] and rsiM[3] <= RSIThreshold and close[3] < lowerBB[3] and bbw[3] > iBBThreshold
back4 = isRed[4] and rsiM[4] <= RSIThreshold and close[4] < lowerBB[4] and bbw[4] > iBBThreshold
back5 = isRed[5] and rsiM[5] <= RSIThreshold and close[5] < lowerBB[5] and bbw[5] > iBBThreshold

for1 = isGreen[1] and rsiM[1] >= RSIDown and close[1] > upperBB[1] and bbw[1] > iBBThreshold
for2 = isGreen[2] and rsiM[2] >= RSIDown and close[2] > upperBB[2] and bbw[2] > iBBThreshold
for3 = isGreen[3] and rsiM[3] >= RSIDown and close[3] > upperBB[3] and bbw[3] > iBBThreshold
for4 = isGreen[4] and rsiM[4] >= RSIDown and close[4] > upperBB[4] and bbw[4] > iBBThreshold
for5 = isGreen[5] and rsiM[5] >= RSIDown and close[5] > upperBB[5] and bbw[5] > iBBThreshold

weGoUp = isGreen and (back1 or back2 or back3 or back4 or back5) and (high > high[1]) and barstate.isc
upThrust = weGoUp and not weGoUp[1] and not weGoUp[2] and not weGoUp[3] and not weGoUp[4]
weGoDown = isRed and (for1 or for2 or for3 or for4 or for5) and (low < low[1]) and barstate.isconfirme
downThrust = weGoDown and not weGoDown[1] and not weGoDown[2] and not weGoDown[3] and not weGoDown[4]

if (upThrust or downThrust) or (upThrust[1] or downThrust[1])
    iTPSignalCount := iTPSignalCount + vTramp

//plotshape(bShowTramp and upThrust ? hl2 : na, title="Trampoline", text="T", location=location.belowb
//plotshape(bShowTramp and downThrust ? hl2 : na, title="Trampoline", text="T", location=location.abov



// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=   Squeeze Relaxer version 2.1  =-=-=-=-=-=-=-=-=-=-=-=-=-

// Average Directional Index
```

```
sqTolerance = input.int(2, title="Squeeze Tolerance (lower = more sensitive)", group="Relaxing Setting
adxSqueezeQ = input.int(21, title="ADX Threshold for TTM Squeeze", group="Relaxing Settings", tooltip=

adxValueQ = adx(dilen, adxlen)
sigabove19Q = adxValueQ > adxSqueezeQ

var cGreen = 0
var cRed = 0
var pos = false
var neg = false

sqlength = 20
multQ = 2.0
lengthKC = 20
multKC = 1.5

useTrueRange = true
source = close
basis = ta.sma(source, sqlength)
dev1 = multKC * ta.stdev(source, sqlength)
upperBBsq = basis + dev1
lowerBBsq = basis - dev1
ma = ta.sma(source, lengthKC)
rangeQ = high - low
rangema = ta.sma(rangeQ, lengthKC)
upperKC = ma + rangema * multKC
lowerKC = ma - rangema * multKC
sqzOn  = (lowerBBsq > lowerKC) and (upperBBsq < upperKC)
sqzOff = (lowerBBsq < lowerKC) and (upperBBsq > upperKC)
noSqz  = (sqzOn == false) and (sqzOff == false)

avg1 = math.avg(ta.highest(high, lengthKC), ta.lowest(low, lengthKC))
avg2 = math.avg(avg1, ta.sma(close, lengthKC))
val = ta.linreg(close - avg2, lengthKC, 0)

pos := false
neg := false

// if squeeze is bright RED, increment by one
if (val < nz(val[1]) and val < 5 and not sqzOn)
    cRed := cRed + 1

// if squeeze is bright GREEN, increment by one
if (val > nz(val[1]) and val > 5 and not sqzOn)
    cGreen := cGreen + 1

// if bright RED squeeze is now dim, momentum has changed.  Is ADX also above 19? - add a marker to ch
if (val > nz(val[1]) and cRed > sqTolerance and val < 5 and not pos[1] and sigabove19 == true)
    cRed := 0
    pos := true

// if bright GREEN squeeze is now dim, momentum has changed.  Is ADX also above 19? - add a marker to
if (val < nz(val[1]) and cGreen > sqTolerance and val > 5 and not neg[1] and sigabove19 == true)
    cGreen := 0
    neg := true
```

```
buySignal1 = pos and barstate.isconfirmed
sellSignal1 = neg and barstate.isconfirmed

if (buySignal1 or sellSignal1) or (buySignal1[1] or sellSignal1[1])
    iTPSignalCount := iTPSignalCount + vSqueeze

//plotshape(bShowSqueeze and pos ? pos : na, title="Squeeze Buy Signal", style=shape.diamond, location
//plotshape(bShowSqueeze and neg ? neg : na, title="Squeeze Sell Signal", style=shape.diamond, locatic

cSQColor = color.white
cSQBorder = color.white
cSQWick = color.white

if val > 0
    if val > nz(val[1])
        cSQColor := color.from_gradient(math.abs(val), 0, 30, color.new(colorBigGreen, 50), color.new(
        cSQBorder := colorBigGreen
        cSQWick := cSQColor
    if val < nz(val[1])
        cSQColor := color.new(colorBigGreen, 70)
        cSQBorder := color.new(color.black, 100)
        cSQWick := cSQColor
else
    if val < nz(val[1])
        cSQColor := color.from_gradient(math.abs(val), 0, 30, color.new(colorBigRed, 50), color.new(cc
        cSQBorder := colorBigRed
        cSQWick := cSQColor
    if val > nz(val[1])
        cSQColor := color.new(colorBigRed, 50)
        cSQBorder := color.new(color.black, 100)
        cSQWick := cSQColor



// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=   VECTOR CANDLES  =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

import TradersReality/Traders_Reality_Lib/2 as trLib

color redVectorColor = colorBigRed
color greenVectorColor = colorBigGreen
color violetVectorColor = input.color(title='Violet',defval=color.fuchsia, inline='vectors', group="Ve
color blueVectorColor = input.color(title='Blue', defval=color.rgb(83, 144, 249), inline='vectors', tc
color regularCandleUpColor = input.color(title='Regular: Up Candle', defval=color.new(#02a433, 99), ir
color regularCandleDownColor = input.color(title='Regular: Down Candle', defval=color.new(#a10101, 99)

bool overrideSym = false
string pvsraSym = ''
bool colorOverride = true

pvsraVolume(overrideSymbolX, pvsraSymbolX, tickerIdX) =>
    request.security(overrideSymbolX ? pvsraSymbolX : tickerIdX, '', [volume,high,low,close,open], bar

[pvsraVolume, pvsraHigh, pvsraLow, pvsraClose, pvsraOpen]  = pvsraVolume(overrideSym, pvsraSym, syminf
```

```
[pvsraColor, alertFlag, averageVolume, volumeSpread, highestVolumeSpread] = trLib.calcPvsra(pvsraVolum

bVectorGreen = pvsraColor == greenVectorColor
bVectorRed = pvsraColor == redVectorColor


// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=   TOTAL RECALL  =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

if (bVectorGreen and (close[0] == upClose[0] or close[1] == upClose[1] or close[2] == upClose[2] or cl
    bGreenSignal := true
//plotshape(bGreenSignal and bShowEarlyReversal and barstate.isconfirmed ? 1 : na, title="Reversal App
//plotcandle(open, high, low, close, "", color=pvsraColor, wickcolor=pvsraColor, bordercolor=color.rgb

if (bVectorRed and (close[0] == downClose[0] or close[1] == downClose[1] or close[2] == downClose[2] c
    bRedSignal := true
//plotshape(bRedSignal and bShowEarlyReversal and barstate.isconfirmed ? 1 : na, title="Reversal Approc
//plotcandle(open, high, low, close, "", color=pvsraColor, wickcolor=pvsraColor, bordercolor=color.rgb

if (bGreenSignal or bRedSignal) or (bGreenSignal[1] or bRedSignal[1])
    iTPSignalCount := iTPSignalCount + vEarlyRev


// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=   THE SHARK  =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

bApply25and75 = input(false, title="Apply 25/75 RSI rule", group="Shark Settings")

ema50 = ta.ema(close, 50)
ema200 = ta.ema(close, 200)
ema400 = ta.ema(close, 400)
ema800 = ta.ema(close, 800)
wapwap = ta.vwap(close)

bTouchedLine = (ema50<high and ema50>low) or (ema200<high and ema200>low) or (ema400<high and ema400>l

basis5 = ta.sma(rsiM, 30)
dev = 2.0 * ta.stdev(rsiM, 30)
upper7 = basis5 + dev
lower7 = basis5 - dev

bBelow25 = rsiM < 26
bAbove75 = rsiM > 74

if not bApply25and75
    bBelow25 := true
    bAbove75 := true

bShowSharkUp = (rsiM < lower7 and bBelow25) and barstate.isconfirmed
bShowSharkDown = (rsiM > upper7 and bAbove75) and barstate.isconfirmed

if (bShowSharkUp or bShowSharkDown) or (bShowSharkUp[1] or bShowSharkDown[1])
    iTPSignalCount := iTPSignalCount + vShark

//plotchar(bShowShark and bShowSharkUp ? hlcc4 : na, char="🦈", title="Shark", location=location.belo
//plotchar(bShowShark and bShowSharkDown ? hlcc4 : na, char="🦈", title="Shark", location=location.ab
```

```
//////////////////////////   ULTIMATE BUY/SELL INDICATOR   ////////////////////////////////////////

showAllMA = false
showBasisPlot = false
showWatchSignals = true

group3 = "Ultimate Buy Sell"
requireWatchSignals = true
watchSignalLookback = input.int(35, title="# of bars back to use Watch Signals", group=group3, step=1,
useSignalWaiting = false
signalWaitPeriod = input.int(5, title="# of bars before signals are allowed ", group=group3, step=1, t

group4 = "Ultimate Buy Sell"
rsiSource = input(close, title="RSI Data Source", group=group4)
rsiLength = input.int(32, title="RSI Length", minval=1, group=group4, tooltip="RSI is not visible, but
rsiMaType = "RMA"
rsiMaType1 = "WMA"
rsiBasisLength = input.int(32, title="RSI Basis Length", minval=1, group=group4)
rsiMultiplier = input.float(2, minval=1, maxval=3, step=0.1, title="RSI Band Multiplier", group=group4
wmaLength = input.int(3, title="Smoothing Length", minval=1, group=group4)
useRsiWatchSignals = input(true, title="RSI Watch Signals", group=group4, tooltip = "If the RSI Crosse

group5 = "Ultimate Buy Sell"
priceBasisLength = input.int(20, title="Price BBand Basis Length", group=group5, tooltip="Sets the len
priceMaType = "SMA"
priceInnerMultiplier = input.float(2, minval=1, maxval=4, step=0.1, title="Price Inner BB Multiplier",
priceOuterMultiplier = input.float(2.5, minval=2, maxval=6, step=0.1, title="Price Outer BB Multiplier
usePriceBandWatchSignals = input(true, title="Bollinger Band Watch Signals", group=group5, tooltip= "I

group6 = "Ultimate Buy Sell"
atrPeriod = input.int(30, title="ATR Period", group=group6)
maPeriod = input.int(10, title="ATR MA Period", group=group6)
atrMult = input.float(1.5, minval=1, step=0.1, title="ATR Band multiplier", group=group6, tooltip = "C
atrMaType = input.string("WMA", title="ATR Moving Average Type", options=["SMA", "EMA", "WMA", "HMA",
useAtrWatchSignals = input(true, title="ATR watch signals", group=group6)

group8 = "Ultimate Buy Sell"
useRsiSignals = input(true, title="RSI crossing Basis", group=group8, tooltip="Uses RSI crossing RSI b
use75Signals = input(true, title="RSI crossing under 75", group=group8, tooltip="Sell signals from cro
use25Signals = input(true, title="RSI crossing over 25", group=group8, tooltip="Buy signals from cross
useRsiMa = input(true, title="Rsi Crossing a Moving Average", group=group8, tooltip="Signals based on
rsiMaLength = input.int(24, title="Length of additional RSI MA", inline="rsiMa", minval=1, group=group
rsiMaType2 = input.string("WMA", title="Moving Average Type", inline="rsiMa", options=["SMA", "EMA", "

filterBuySell = true
group11 = "Ultimate Buy Sell"
fast_length = input.int(12, title="Fast Length", group=group11, tooltip="Set the fast length for the M
slow_length = input.int(26, title="Slow Length", group=group11, tooltip="Set the slow length for the M
signal_length = input.int(title="Smoothing", minval=1, maxval=50, defval=9, group=group11, tooltip="Se
sma_source = input.string(title="MACD Line MA Type", defval="EMA", options=["SMA", "EMA", "WMA", "HMA"
sma_signal = input.string(title="Signal Line MA Type", defval="EMA", options=["SMA", "EMA", "WMA", "HM

// ATR with bands
```

```
atrMa(src, Length, type) =>
    switch type
        "SMA" => ta.sma(src, Length)
        "EMA" => ta.ema(src, Length)
        "WMA" => ta.wma(src, Length)
        "HMA" => ta.hma(src, Length)
        "VWMA" => ta.vwma(src, Length)
        "RMA" => ta.rma(src, Length)

atrValue = ta.atr(atrPeriod)
atrMaValue = atrMa(close, maPeriod, atrMaType)

upperAtrBand = atrMaValue + atrValue * atrMult
middleAtrBand = atrMaValue
lowerAtrBand = atrMaValue - atrValue * atrMult

bbUpper = atrMaValue + atrValue + atrMult * ta.stdev(close, maPeriod)
bbLower = atrMaValue - atrValue - atrMult * ta.stdev(close, maPeriod)

maType(src, length, type) =>
    switch type
        "SMA" => ta.sma(src, length)
        "EMA" => ta.ema(src, length)
        "WMA" => ta.wma(src, length)
        "HMA" => ta.hma(src, length)
        "VWMA" => ta.vwma(src, length)
        "RMA" => ta.rma(src, length)
fast_ma := maType(close, fast_length, sma_source)
slow_ma := maType(close, slow_length, sma_source)
macd := fast_ma - slow_ma
signal := maType(macd, signal_length, sma_signal)
hist := macd - signal

up := ta.rma(math.max(ta.change(rsiSource), 0), rsiLength)
down := ta.rma(-math.min(ta.change(rsiSource), 0), rsiLength)
rsi := down == 0 ? 100 : up == 0 ? 0 : 100 - (100 / (1 + up / down))

rsiMa(source, length, type) =>
    switch type
        "SMA" => ta.sma(source, length)
        "EMA" => ta.ema(source, length)
        "WMA" => ta.wma(source, length)
        "HMA" => ta.hma(source, length)
        "VWMA" => ta.vwma(source, length)
        "RMA" => ta.rma(source, length)

rsiBasis = rsiMa(rsi, rsiBasisLength, rsiMaType1)
rsiDeviation = ta.stdev(rsi, rsiBasisLength)

upperRsi = rsiBasis + rsiMultiplier * rsiDeviation
lowerRsi = rsiBasis - rsiMultiplier * rsiDeviation

rsiMa = rsiMa(rsi, rsiMaLength, rsiMaType2)

priceMa(src, Length, type) =>
```

```
    switch type
        "SMA" => ta.sma(src, Length)
        "EMA" => ta.ema(src, Length)
        "WMA" => ta.wma(src, Length)
        "HMA" => ta.hma(src, Length)
        "VWMA" => ta.vwma(src, Length)
        "RMA" => ta.rma(src, Length)

calculateBollingerBands(src, priceBasisLength, priceInnerMultiplier, priceOuterMultiplier, priceMaType
    priceBasis = priceMa(src, priceBasisLength, priceMaType)
    priceInnerDeviation = priceInnerMultiplier * ta.stdev(src, priceBasisLength)
    priceOuterDeviation = priceOuterMultiplier * ta.stdev(src, priceBasisLength)
    [priceBasis, priceBasis + priceInnerDeviation, priceBasis - priceInnerDeviation, priceBasis + pric

[priceBasis, upperPriceInner, lowerPriceInner, upperPriceOuter, lowerPriceOuter] = calculateBollingerB

priceCrossOverInner = ta.crossover(close, lowerPriceInner) // Price over outer band
priceCrossUnderInner = ta.crossunder(close, upperPriceInner) // Price under outer band

rsiCrossOverLower = ta.crossover(rsi, lowerRsi) // RSI over lower band
rsiCrossUnderUpper = ta.crossunder(rsi, upperRsi) // RSI under upper band

rsiCrossOverBasis = ta.crossover(rsi, rsiBasis)
rsiCrossUnderBasis = ta.crossunder(rsi, rsiBasis)

rsiCrossOverMa = ta.crossover(rsi, rsiMa)
rsiCrossUnderMa = ta.crossunder(rsi, rsiMa)

rsiCrossUnder75 = ta.crossunder(rsi, 75) // RSI crossunder 75
rsiCrossUnder70 = ta.crossunder(rsi, 70) // RSI crossunder 70
rsiCrossUnder50 = ta.crossunder(rsi, 50) // RSI crossover 50
rsiCrossOver50 = ta.crossover(rsi, 50) // RSI crossover 50
rsiCrossOver30 = ta.crossover(rsi, 30) // RSI crossover 30
rsiCrossOver25 = ta.crossover(rsi, 25) // RSI crossover 25

priceCrossOverBasis = ta.crossover(close, priceBasis)
priceCrossUnderBasis = ta.crossunder(close, priceBasis)

macdBuy = ta.crossover(macd, signal)
macdSell = ta.crossunder(macd, signal)

highUnderAtrLower = ta.crossunder(high, lowerAtrBand)
lowOverAtrUpper = ta.crossover(low, upperAtrBand)

watchesInsideYellowRsi = false
buyAndSellInsideYellowRsi = false

var bool bought = false
var bool sold = false
var bool signalsBlocked = false
var int[] buyWatchArray = array.new_int(na)
var int[] sellWatchArray = array.new_int(na)
var int lastSignalBarIndex = na
bool plotBuy = false
bool plotSell = false
```

```
bool plotBuyBG = false
bool plotSellBG = false

buyWatch1 = (usePriceBandWatchSignals) and (priceCrossOverInner and not rsiCrossOverLower) and (barsta
buyWatch2 = (useRsiWatchSignals) and (rsiCrossOverLower and not priceCrossOverInner) and (barstate.isc
buyWatch3 = (usePriceBandWatchSignals) and (priceCrossOverInner and rsiCrossOverLower) and (barstate.i
buyWatch4 = (usePriceBandWatchSignals) and (priceCrossOverInner) and (barstate.isconfirmed) and (not w
buyWatch5 = (useRsiWatchSignals) and (rsiCrossOverLower) and (barstate.isconfirmed) and (not watchesIr
buyWatch6 = (useRsiWatchSignals) and (rsiCrossOver25) and (barstate.isconfirmed) and (not watchesInsic
buyWatch7 = (useAtrWatchSignals and highUnderAtrLower) and (barstate.isconfirmed) and (not watchesInsi

sellWatch1 = (usePriceBandWatchSignals) and (priceCrossUnderInner and not rsiCrossUnderUpper) and (bar
sellWatch2 = (useRsiWatchSignals) and (rsiCrossUnderUpper and not priceCrossUnderInner) and (barstate.
sellWatch3 = (usePriceBandWatchSignals) and (priceCrossUnderInner and rsiCrossUnderUpper) and (barstat
sellWatch4 = (usePriceBandWatchSignals) and (priceCrossUnderInner) and (barstate.isconfirmed) and (not
sellWatch5 = (useRsiWatchSignals) and (rsiCrossUnderUpper) and (barstate.isconfirmed) and (not watches
sellWatch6 = (useRsiWatchSignals) and (rsiCrossUnder75) and (barstate.isconfirmed) and (not watchesIns
sellWatch7 = (useAtrWatchSignals) and (lowOverAtrUpper) and (barstate.isconfirmed) and (not watchesIns

bool buyWatched = buyWatch1 or buyWatch2 or buyWatch3 or buyWatch4 or buyWatch5 or buyWatch6 or buyWat
bool sellWatched = sellWatch1 or sellWatch2 or sellWatch3 or sellWatch4 or sellWatch5 or sellWatch6 or

array.push(buyWatchArray, buyWatched ? 1 : na)
array.push(sellWatchArray, sellWatched ? 1 : na)

while array.size(buyWatchArray) > watchSignalLookback
    array.shift(buyWatchArray)
while array.size(sellWatchArray) > watchSignalLookback
    array.shift(sellWatchArray)

buyWatchSumMet = (array.sum(buyWatchArray) >= 1)
sellWatchSumMet = (array.sum(sellWatchArray) >= 1)

buyWatchMet = (buyWatchSumMet)
sellWatchMet = (sellWatchSumMet)

combinedBuySignals = rsiCrossOverBasis or rsiCrossOver25 or rsiCrossOverMa // or buySignal7 or buySigr
combinedSellSignals = rsiCrossUnderBasis or rsiCrossUnder75 or rsiCrossUnderMa // or sellSignal7 or se

buySignals = ((not requireWatchSignals and combinedBuySignals) or (requireWatchSignals and buyWatchMet
sellSignals = ((not requireWatchSignals and combinedSellSignals) or (requireWatchSignals and sellWatch

if (buySignals) and (not buyAndSellInsideYellowRsi) and (not buyWatched) and (not signalsBlocked)
    plotBuyBG := true
else if (sellSignals) and (not buyAndSellInsideYellowRsi) and (not sellWatched) and (not signalsBlocke
    plotSellBG := true
else
    plotBuyBG := false
    plotSellBG := false

if (buySignals) and (barstate.isconfirmed) and (not buyAndSellInsideYellowRsi) and (not buyWatched) ar
    bought := true
    sold := false
    plotBuy := true
    lastSignalBarIndex := bar_index
```

```
    array.clear(buyWatchArray)
    array.clear(sellWatchArray)
else if (sellSignals) and (barstate.isconfirmed) and (not buyAndSellInsideYellowRsi) and (not sellWatc
    sold := true
    bought := false
    plotSell := true
    lastSignalBarIndex := bar_index
    array.clear(sellWatchArray)
    array.clear(buyWatchArray)
else
    plotBuy := false
    plotSell := false

//plotshape(bBuySell and plotBuy and trendUpWAE > 0 ? true : na, title="BUY/LONG", color=color.new(col
//plotshape(bBuySell and plotSell and trendDownWAE > 0 ? true : na, title="SELL/SHORT", color=color.ne




// ====================================   VODKA SHOT   ============================================

// ==================================  NEGLECTED VOL by DGT  ======================================--

nzVolume = nz(volume)
source5   = barstate.isconfirmed ? close : close[1]
vsource  = nzVolume ? barstate.isconfirmed ? ta.obv : ta.obv[1] : na
corr     = ta.correlation(source5, vsource, 14)
volAvgS  = ta.sma(nzVolume, 14)
volAvgL  = ta.sma(nzVolume, 14 * 5)
volDev   = (volAvgL + 1.618034 * ta.stdev(volAvgL, 14 * 5)) / volAvgL * 11 / 100
volRel   = nzVolume / volAvgL
momentum = ta.change(vsource, 14) / 14
momOsc   = ta.linreg(momentum / volAvgS * 1.618034, 5, 0)

vbcbColor = if close < open
    if nzVolume > volAvgS * 1.618034
        #910000
    else if nzVolume >= volAvgS * .618034 and nzVolume <= volAvgS * 1.618034
        color.red
    else
        color.orange
else
    if nzVolume > volAvgS * 1.618034
        #006400
    else if nzVolume >= volAvgS * .618034 and nzVolume <= volAvgS * 1.618034
        color.green
    else
        #7FFFD4

bColor5 = color.new(color.black, 25)
gColor = color.new(color.gray, 50)

// ==================================  RedK Dual VADER with Energy Bars [VADER-DEB]  ================

f_derma(_data, _len, MAOption) =>
```

```
    value =
      MAOption == 'SMA' ? ta.sma(_data, _len) :
      MAOption == 'EMA' ? ta.ema(_data, _len) :
      ta.wma(_data, _len)

rlength  = input.int(12, minval=1)
DER_avg = input.int(5, 'Average', minval=1, inline='DER', group='Directional Energy Ratio')
MA_Type5 = input.string('WMA', 'DER MA type', options=['WMA', 'EMA', 'SMA'], inline='DER', group='Dire
rsmooth  = input.int(3, 'Smooth', minval=1,  inline='DER_1', group='Directional Energy Ratio')

show_senti = input.bool(true, 'Sentiment',  inline='DER_s', group='Directional Energy Ratio')
senti   = input.int(20, 'Length', minval=1, inline='DER_s', group='Directional Energy Ratio')

v_calc  = input.string('Relative', 'Calculation', options=['Relative', 'Full', 'None'], group='Volume
vlookbk = input.int(20, 'Lookback (for Relative)', minval=1,                            group='Volume

v5 = volume

vola    =
  v_calc == 'None' or na(volume) ? 1 :
  v_calc == 'Relative' ?   ta.stoch(v5, v5, v5, vlookbk) / 100 :
  v5

R       = (ta.highest(2) - ta.lowest(2)) / 2                  // R is the 2-bar average bar range -
sr      = ta.change(close) / R                                // calc ratio of change to R
rsr     = math.max(math.min(sr, 1), -1)                       // ensure ratio is restricted to +1/-1
c       = fixnan(rsr * vola)
c_plus  = math.max(c, 0)                                      // calc directional vol-accel energy
c_minus = -math.min(c, 0)
avg_vola    = f_derma(vola, rlength, MA_Type5)
dem         = f_derma(c_plus, rlength, MA_Type5)  / avg_vola         // directional energy ratio
sup         = f_derma(c_minus, rlength, MA_Type5) / avg_vola
adp         = 100 * ta.wma(nz(dem), DER_avg)                      // average DER
asp         = 100 * ta.wma(nz(sup), DER_avg)
anp         = adp - asp                                           // net DER..
anp_s       = ta.wma(anp, rsmooth)
s_adp       = 100 * ta.wma(nz(dem), senti)                          // average DER for sentiment ler
s_asp       = 100 * ta.wma(nz(sup), senti)
V_senti     = ta.wma(s_adp - s_asp, rsmooth)
c_adp   = color.new(#11ff20, 30)
c_asp   = color.new(#ff1111, 30)
c_fd    = color.new(color.green, 80)
c_fs    = color.new(color.red, 80)
c_zero  = color.new(#ffee00, 70)
c_up5    = color.new(#11ff20, 0)
c_dn5    = color.new(#ff1111, 0)

up5      = anp_s >= 0
s_up    = V_senti >= 0

c_grow_above = #1b5e2080
c_grow_below = #dc4c4a80
c_fall_above = #66bb6a80
c_fall_below = #ef8e9880
```

```
sflag_up = math.abs(V_senti) >= math.abs(V_senti[1])
bo = fixnan(asp)
bc = fixnan(adp)
bh = math.max(bo, bc)
bl = math.min(bo, bc)

rising      = ta.change(bc) > 0
c_barup     = #11ff2088
c_bardn     = #ff111188
c_bardj     = #ffffff88

barcolor    = bc > bo and rising ? c_barup : bc < bo and not rising ? c_bardn : c_bardj

// ==================================  RedK Slow_Smooth WMA, RSS_WMA v3 ============================

f_LazyLine(_data, _length) =>
    w1 = 0, w2 = 0, w3 = 0
    L1 = 0.0, L2 = 0.0, L3 = 0.0
    w = _length / 3

    if _length > 2
        w2 := math.round(w)
        w1 := math.round((_length-w2)/2)
        w3 :=   int((_length-w2)/2)

        L1 := ta.wma(_data, w1)
        L2 := ta.wma(L1, w2)
        L3 := ta.wma(L2, w3)
    else
        L3 := _data
    L3

LL = f_LazyLine(close, 21)

lc_up        = color.new(#33ff00, 0)
lc_dn        = color.new(#ff1111, 0)
luptrend     = LL > LL[1]
SigMulti    = 1.0

SignalOn    = barstate.isconfirmed
SwingDn     = luptrend[1] and not(luptrend) and barstate.isconfirmed
SwingUp     = luptrend and not(luptrend[1]) and barstate.isconfirmed

dl = SigMulti / 100 * LL

upwards = LL > LL[1] and (barcolor == c_barup) and up and (open < close and volRel * .145898 > volDev)
downwards = LL < LL[1]  and (barcolor == c_bardn) and (open > close and volRel * .145898 > volDev)

pBuyVodka = upwards and not upwards[1] and not upwards[2] and not upwards[3] and not upwards[4]
pSellVodka = downwards and not downwards[1] and not upwards[2] and not upwards[3] and not upwards[4]


// ====================================   JOHN WICK SETTINGS  ======================================

upWick50PercentLarger = close > open and math.abs(high - close) > math.abs(open - close)
```

```
downWick50PercentLarger = close < open and math.abs(low - close) > math.abs(open - close)

wlengthBB = input.int(20, minval=1, group="Wicking Bollinger Bands")
wsrcBB = input(close, title="Source", group="Wicking Bollinger Bands")
wmultBB = input.float(2.5, minval=0.001, maxval=50, title="StdDev", group="Wicking Bollinger Bands")
woffsetBB = input.int(0, "Offset", minval = -500, maxval = 500, group="Wicking Bollinger Bands")
wbasisBB = ta.sma(wsrcBB, wlengthBB)
wdevBB = wmultBB * ta.stdev(wsrcBB, wlengthBB)
wupperBB = wbasisBB + wdevBB
wlowerBB = wbasisBB - wdevBB

bBBUp = low <= wlowerBB and close >= wlowerBB and close < open
bBBDown = high >= wupperBB and close < wupperBB and close > open

if (bBBUp or bBBDown) or (bBBUp[1] or bBBDown[1])
    iTPSignalCount := iTPSignalCount + vBands


// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=  Rational Quadratic Kernel Estimate  =-=-=-=-=-=-=-=-=-=-=-=-

h = input.float(21., 'Lookback Window', minval=3., group='Rational Quadratic Kernel Estimate')
r = input.float(8., 'Relative Weighting', step=0.25, group='Rational Quadratic Kernel Estimate')
x_0 = input.int(15, "Start Regression at Bar", group='Rational Quadratic Kernel Estimate')

size = array.size(array.from(close))

kernel_regression(_src, _size, _h) =>
    float _currentWeight = 0.
    float _cumulativeWeight = 0.
    for i = 0 to _size + x_0
        y = _src[i]
        w = math.pow(1 + (math.pow(i, 2) / ((math.pow(_h, 2) * 2 * r))), -r)
        _currentWeight += y*w
        _cumulativeWeight += w
    _currentWeight / _cumulativeWeight

yhat1 = kernel_regression(close, size, h)


// plot(yhat1, "Rational Quadratic Kernel Estimate", color=plotColor, linewidth=2)


// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=  RSI  =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //

Rsi2 = ta.rsi(close, 14)
//    if upWick50PercentLarger and Rsi2 > rsiUpperW
//    if downWick50PercentLarger and Rsi2 < rsiLowerW


// =-=-=-=-=-=-=-=-=-=-=-=   TIDAL WAVE   =-=-=-=-=-=-=-=-=-=-=-=-=-= //
// IMPORTANT: Credit to Aaron D for all ideas in this indicator
// Go subscribe to him at https://www.youtube.com/@aarond98

var line[] ll = array.new_line()
const int upTrend = 1
```

```
const int downTrend = 2
var int waveState = na

redCandle = close < open
greenCandle = close > open
iBodyWidth = math.abs(close - open)
bDoji = iBodyWidth <= iMaxBody and bIgnoreDoji
noOverlapRed = false
noOverlapGreen = false

brightGreen = false
brightRed = false

gapGreen = false
gapRed = false
bNewGap = false

for [index, line] in ll
    if (high > line.get_y1(line) and low < line.get_y1(line))
        line.delete(array.get(ll, index))

if greenCandle and not bDoji and barstate.isconfirmed
    for i = 1 to 200
        if (brightRed[i]) // if bright red candle, stop
            break
        else if (waveState==upTrend and redCandle[i]) // if we're in a uptrend, and the candle is red,
            break
        else if (waveState==downTrend and open >= close[i] and greenCandle[i])
            noOverlapGreen := true
            brightGreen := true
            waveState := upTrend
            break
    if (open >= close[1] and greenCandle[1])
        if bTrackBar
            array.push(ll, line.new(bar_index, open, bar_index + iBarExtend, open, color=color.new(col
        waveState := upTrend
        brightGreen := true
        gapGreen := true

if redCandle and not bDoji and barstate.isconfirmed
    for i = 1 to 200
        if (brightGreen[i]) // if bright green candle, stop
            break
        else if (waveState==downTrend and greenCandle[i]) // if we're in a downtrend, and the candle i
            break
        else if (waveState==upTrend and open <= close[i] and redCandle[i])
            noOverlapRed := true
            brightRed := true
            waveState := downTrend
            break
    if (redCandle[1] and open < close[1])
        if bTrackBar
            array.push(ll, line.new(bar_index, open, bar_index + iBarExtend, open, color=color.new(col
        waveState := downTrend
        brightRed := true
```

```
        gapRed := true

bWaddahGreen = trendUpWAE > 0
bWaddahRed = trendDownWAE > 0

bBigBuy1 = plotBuy or plotBuy[1] or plotBuy[2] or plotBuy[3]
bBigSell1 = plotSell or plotSell[1] or plotSell[2] or plotSell[3]

cloudTrans = math.round(math.abs(FanVMA - mg))


// ==========================   HEMA  ================================

alphaLength = input.int(title="Alpha Length", defval=20, minval=1, group="HEMA settings")
gammaLength = input.int(title="Gamma Length", defval=20, minval=1, group="HEMA settings")
highlightMovements = true
src = input.source(title="Source", defval=close, group="HEMA settings")

alpha = 2 / (alphaLength + 1)
gamma = 2 / (gammaLength + 1)

bH = 0.
hema = 0.

hema := (1 - alpha) * (nz(hema[1]) + nz(bH[1], src)) + alpha * src
bH := (1 - gamma) * nz(bH[1]) + gamma * (hema - nz(hema[1]))

hemaColor = highlightMovements ? (hema > hema[1] ? color.new(colorBigGreen, 40) : color.new(colorBigRe

bSkullUp = not bShowQuad ? ta.crossover(ema9, ema21) and bShow921 : ta.crossover(ema9, yhat1) and bShc
bSkullDown = not bShowQuad ? ta.crossunder(ema9, ema21) and bShow921 : ta.crossunder(ema9, yhat1) and


// ==========================   CLOUD RENDER  ================================

color cColor = na

candleInCloud = FanVMA > mg ? (close < FanVMA or open < FanVMA) and (close > mg or open > mg) : (close

if FanVMA > mg
    cColor := color.from_gradient(iSource, MID_CLOUD, MAX_CLOUD, color.new(colorBigGreen, iLowCloud),
    if (cloudType=="Simple")
        cColor := color.new(colorBigGreen, iSimpleCloud)
else
    cColor := color.from_gradient(iSource, MIN_CLOUD, MID_CLOUD, color.new(colorBigRed, iLowCloud), cc
    if (cloudType=="Simple")
        cColor := color.new(colorBigRed, iSimpleCloud)

lineMD = plot(cloudType != "None" ? mg : na, title="", color=color.new(color.blue, 100), title="McGinl
FVMA = plot(cloudType != "None" ? FanVMA : na, color=color.new(color.white, 100), title="Bixord FVMA")

fill(FVMA, lineMD, color=cColor, title="Cloud")


// =-=-=-=-=-=-=-=-=-=-=-=-=   Breaks and Retests [HG]   =-=-=-=-=-=-=-=-=-=-=-=-= //
```

```
bb              = 20
rTon            = true
rTcc            = false
rThv            = false
breakText       = 'Break'

pl = fixnan(ta.pivotlow(low, bb, bb))
ph = fixnan(ta.pivothigh(high, bb, bb))

s_yLoc = low[bb + 1] > low[bb - 1] ? low[bb - 1] : low[bb + 1]
r_yLoc = high[bb + 1] > high[bb - 1] ? high[bb + 1] : high[bb - 1]

drawBox(condition, y1, y2, color) =>
    var box drawBox = na
    if condition
        drawBox := box.new(bar_index - bb, y1, bar_index, y2, color.new(color, 100), bgcolor = color.r
    [drawBox]

breakLabel(y, color, style, textform) => label.new(bar_index, y, textform, textcolor = color, style =
retestCondition(breakout, condition) => ta.barssince(na(breakout)) > 2 and condition
repaint(c1, c2, c3) => rTon ? c1 : rThv ? c2 : rTcc ? c3 : na

[sBox] = drawBox(ta.change(pl), s_yLoc, pl, color.red)
[rBox] = drawBox(ta.change(ph), ph, r_yLoc, color.red)
sTop = box.get_top(sBox), rTop = box.get_top(rBox)
sBot = box.get_bottom(sBox), rBot = box.get_bottom(rBox)

var bool sBreak = na
var bool rBreak = na
cu = repaint(ta.crossunder(close, box.get_bottom(sBox)), ta.crossunder(low, box.get_bottom(sBox)), ta.
co = repaint(ta.crossover(close, box.get_top(rBox)), ta.crossover(high, box.get_top(rBox)), ta.crossov

switch
    cu and na(sBreak) =>
        sBreak := true
    co and na(rBreak) =>
        rBreak := true

s1 = retestCondition(sBreak, high >= sTop and close <= sBot)
s2 = retestCondition(sBreak, high >= sTop and close >= sBot and close <= sTop)
s3 = retestCondition(sBreak, high >= sBot and high <= sTop)
s4 = retestCondition(sBreak, high >= sBot and high <= sTop and close < sBot)
r1 = retestCondition(rBreak, low <= rBot and close >= rTop)
r2 = retestCondition(rBreak, low <= rBot and close <= rTop and close >= rBot)
r3 = retestCondition(rBreak, low <= rTop and low >= rBot)
r4 = retestCondition(rBreak, low <= rTop and low >= rBot and close > rTop)

retestEvent(c1, c2, c3, c4, y1, y2, col, style, pType) =>
    if true
        var bool retOccurred = na
        retActive   = c1 or c2 or c3 or c4
        retEvent    = retActive and not retActive[1]
        retValue    = ta.valuewhen(retEvent, y1, 0)
```

```
        if pType == 'ph' ? y2 < ta.valuewhen(retEvent, y2, 0) : y2 > ta.valuewhen(retEvent, y2, 0)
            retEvent := retActive

        retValue := ta.valuewhen(retEvent, y1, 0)
        retSince = ta.barssince(retEvent)
        var retLabel = array.new<label>()

        if retEvent and candleInCloud and bShowRetest
            retOccurred := na
            array.push(retLabel, label.new(bar_index - retSince, y2[retSince], text = '!', color = col

        if array.size(retLabel) == 2
            label.delete(array.first(retLabel))
            array.shift(retLabel)

        retConditions = pType == 'ph' ? repaint(close >= retValue, high >= retValue, close >= retValue
        retValid = ta.barssince(retEvent) > 0 and ta.barssince(retEvent) <= 2 and retConditions and nc

        if retValid and candleInCloud and bShowRetest
            label.new(bar_index - retSince, y2[retSince], text = '!', color = color.new(#d900ff, 50),
            retOccurred := true

        if (retValid or ta.barssince(retEvent) > 2) and array.size(retLabel) > 0
            label.delete(array.first(retLabel))

        if pType == 'ph' and ta.change(ph) and retOccurred
            retOccurred := na

        if pType == 'pl' and ta.change(pl) and retOccurred
            retOccurred := na
        [retValid, retEvent, retValue]

[rRetValid, rRetEvent] = retestEvent(r1, r2, r3, r4, high, low, color.red, label.style_label_upper_lef
[sRetValid, sRetEvent] = retestEvent(s1, s2, s3, s4, low, high, color.red, label.style_label_lower_lef



// ==========================   PLOTS   ================================

buyChar = noOverlapGreen and waveState[1]==downTrend
sellChar = noOverlapRed and waveState[1]==upTrend

plotchar(not bBigBuy1 and buyChar ? 1 : na, char="①", location=location.belowbar, color=color.rgb(0, 2
plotchar(not bBigSell1 and sellChar ? 1 : na, char="①", location=location.abovebar, color=color.rgb(25

plotchar(bBigBuy1 and buyChar ? 1 : na, char="❶", location=location.belowbar, color=color.rgb(0, 255,
plotchar(bBigSell1 and sellChar ? 1 : na, char="❶", location=location.abovebar, color=color.rgb(255,

plotchar(bShowPlus and gapGreen and waveState[1]==upTrend ? 1 : na, char="+", location=location.belowb
plotchar(bShowPlus and gapRed and waveState[1]==downTrend ? 1 : na, char="+", location=location.aboveb

plotchar(bShowBigPlus and pBuyVodka and waveState[1]==upTrend, char="✚", location=location.belowbar,
plotchar(bShowBigPlus and pSellVodka and waveState[1]==downTrend, char="✚", location=location.aboveba

plotchar(bShowProfit and iTPSignalCount >= iMaxProfit and waveState==upTrend, char="✓", location=loca
```

```
plotchar(bShowProfit and iTPSignalCount >= iMaxProfit and waveState==downTrend, char="✓", location=lo

plotchar(bShowProfit and iTPSignalCount >= iProfit and waveState==upTrend, char="✓", location=locatio
plotchar(bShowProfit and iTPSignalCount >= iProfit and waveState==downTrend, char="✓", location=locat

plotshape(bSkullDown, style=shape.triangledown, location=location.abovebar, size=size.tiny, color=colo
plotshape(bSkullUp, style=shape.triangleup, location=location.belowbar, size=size.tiny, color=color.ne

plot(bShowHEMA ? hema : na, title="HEMA", linewidth=1, color=color.new(color.fuchsia, 20))

ThreeOutUp = redCandle[2] and greenCandle[1] and greenCandle and open[1] < close[2] and open[2] < clos
ThreeOutDown = greenCandle[2] and redCandle[1] and redCandle and open[1] > close[2] and open[2] > clos
if (ThreeOutDown or ThreeOutUp) and bShowRevPattern
    //cBodyColor := color.rgb(255, 230, 0)
    cWickColor := color.rgb(255, 230, 0)
    cBorderColor := color.rgb(255, 230, 0)

plotcandle(sCandleType=="Vector" ? open : na, high, low, close, color=pvsraColor, wickcolor=pvsraColor
plotcandle(sCandleType=="Volume Delta" ? open : na, high, low, close, color=cCVDColor, wickcolor=cCVDW
plotcandle(sCandleType=="Waddah" ? open : na, high, low, close, "", color=cBodyColor, wickcolor=cWickC
plotcandle(sCandleType=="Squeeze" ? open : na, high, low, close, color=cSQColor, wickcolor=cSQWick, bc

//plotshape(ThreeOutUp, style=shape.flag, location=location.abovebar, size=size.tiny, color=color.gree
//plotshape(ThreeOutDown, style=shape.flag, location=location.belowbar, size=size.tiny, color=color.re

//plotcandle(open, high, low, close, "", color=color.white, wickcolor=color.white, bordercolor=color.w
//plotcandle(open, high, low, close, "", color=color.white, wickcolor=color.white, bordercolor=color.w

// ==========================   ALERTS  ================================

alertcondition(plotBuy or plotSell, "Ultimate Buy/Sell Signal", "Ultimate Buy/Sell Signal")
alertcondition(not bBigBuy1 and noOverlapGreen and waveState[1]==downTrend, "Buy Signal Basic", "Buy S
alertcondition(not bBigSell1 and noOverlapRed and waveState[1]==upTrend, "Sell Signal Basic", "Sell Si
alertcondition(bBigBuy1 and noOverlapGreen and waveState[1]==downTrend, "Buy Signal Super", "Buy Signa
alertcondition(bBigSell1 and noOverlapRed and waveState[1]==upTrend, "Sell Signal Super", "Sell Signal
alertcondition((gapGreen and waveState[1]==upTrend) or (gapRed and waveState[1]==downTrend), "Small Pl
alertcondition((pBuyVodka and waveState[1]==upTrend) or (pSellVodka and waveState[1]==downTrend), "Lar
alertcondition(iTPSignalCount >= iMaxProfit, "Take Partial Profit", "Take Partial Profit")
alertcondition(iTPSignalCount >= iProfit, "Take FULL Profit", "Take FULL Profit")
alertcondition(bSkullDown or bSkullUp, "9/21 EMA Cross", "9/21 EMA Cross")
```

PDF document made with CodePrint using Prism